

Android Testing

Android Testing: generalità

Un'applicazione Android in senso lato è composta di un lato client e di una o più tipologie di risorse lato server

Web Applications, Web Services, Risorse REST ...

Ci si limiterà allo studio delle problematiche del testing della parte client, la cosiddetta app

Una app Android è sostanzialmente un'applicazione interattiva sottoposta a:

Eventi utente (eventi touch, segnali da sensori)

Eventi di sistema (interruzioni, segnali broadcast)

Android Testing: generalità

E' necessario definire, adattandoli all'ambiente Android:

test models, per rappresentare le tipologie di elementi e interazioni da considerare e procurare;

testing levels, che specifichino i diversi punti di vista e obiettivi rispetto ai quali viene progettato il testing;

test strategies, che definiscono obiettivi, euristiche e algoritmi da seguire nella progettazione dei casi di test;

testing processes, che definiscono le modalità di esecuzione dei processi per il testing delle applicazioni Android;

testing tools, strumenti a supporto delle attività di testing, in particolare a supporto della loro automazione

Unit Testing

Che cosa è possibile considerare come unità, nel testing di un'applicazione Android?

Activity

Service

Broadcast Receiver

Content Provider

Classi Java semplici, che non estendono classi del framework Android

Testing con JUnit

E' possibile utilizzare un framework come Junit per le applicazioni Android (in particolare per le Activity)?

Problema: una sola activity può accedere attivamente all'interfaccia utente.

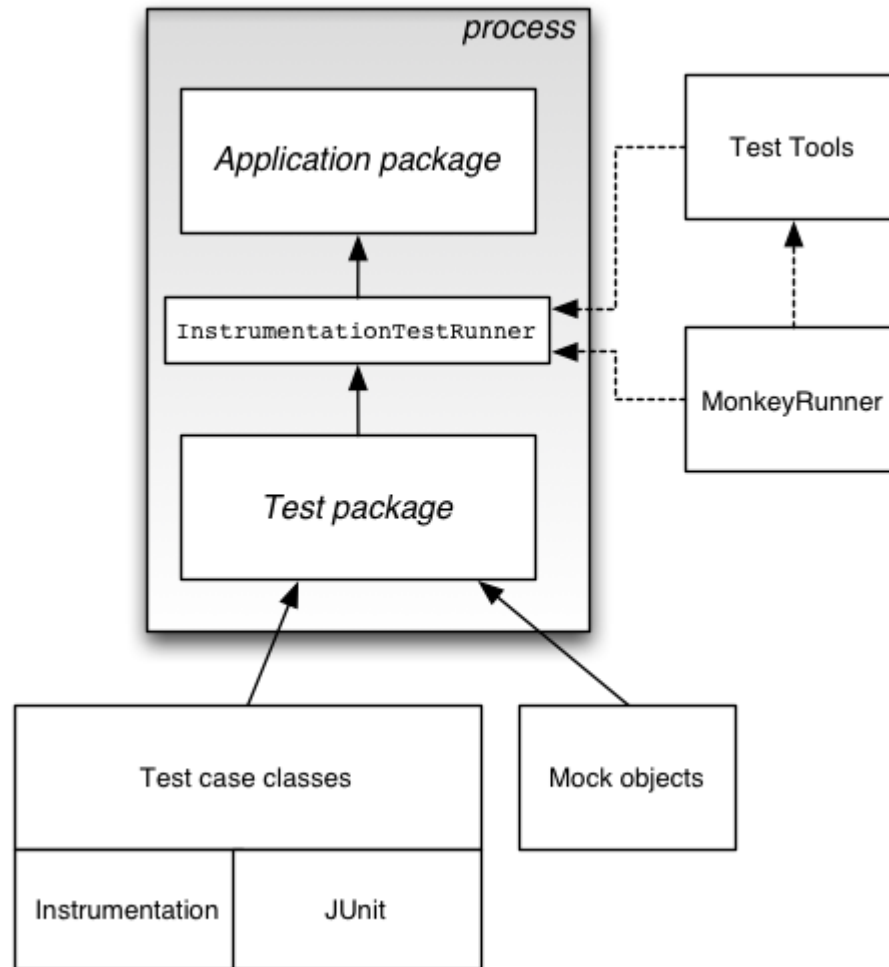
Soluzione: un framework di testing che esegua la activity sotto testing come sua parte, utilizzando funzionalità di strumentazione per poterla monitorare

In questo modo, è possibile pensare di testare una Activity scrivendo dei classici Android JUnit Test Cases, nell'ambito, però, di un project separato

Da notare che il progetto di test e il progetto da testare devono avere la stessa signature (di solito si usa quella pubblica di debug)

http://developer.android.com/guide/topics/testing/activity_testing.html

Android Test Architecture

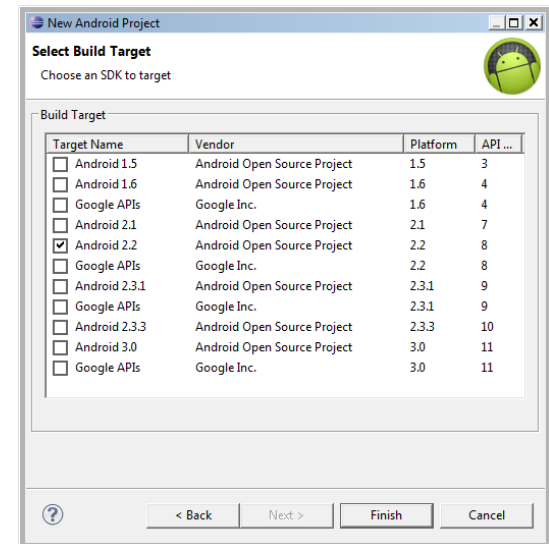
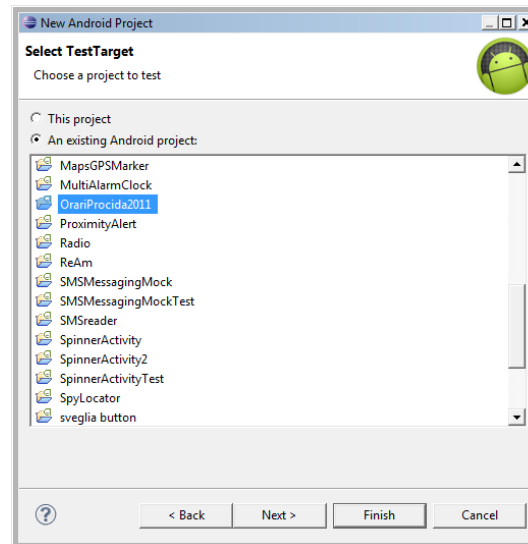
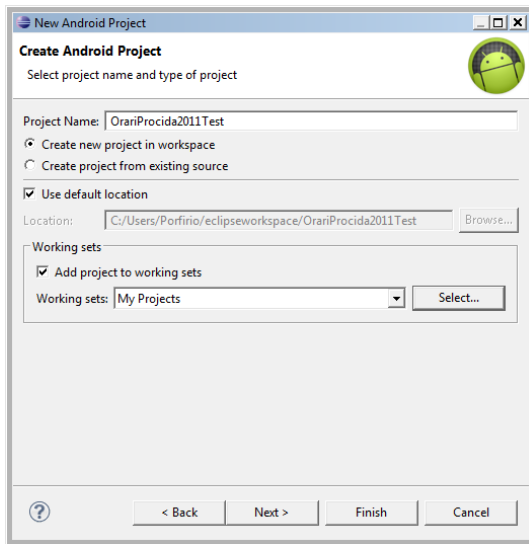


Creazione di un progetto di test

Da linea di comando si può scrivere

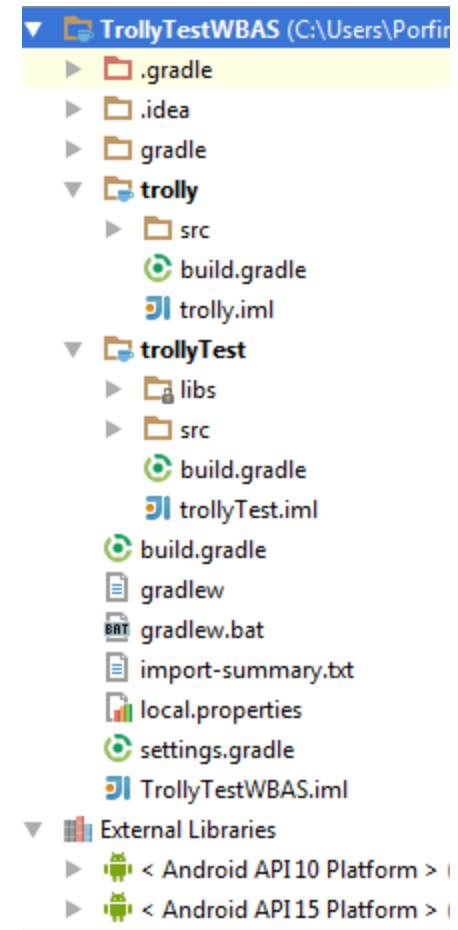
`android create test-project -m <main_path> -n <project_name> -p <test_path>`

In Eclipse è sufficiente utilizzare il widget di creazione progetto



Creazione di un progetto di test

Con Android Studio i nuovi progetti sono configurati per poter eseguire casi di test scritti (ad esempio) in un package chiamato test oppure in un altro progetto di test, che viene affiancato al progetto di sviluppo e aperto nello stesso progetto Android Studio



Manifest di un progetto di test

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.porfirio.orariprocida2011.test"
    android:versionCode="1"
    android:versionName="1.0" >

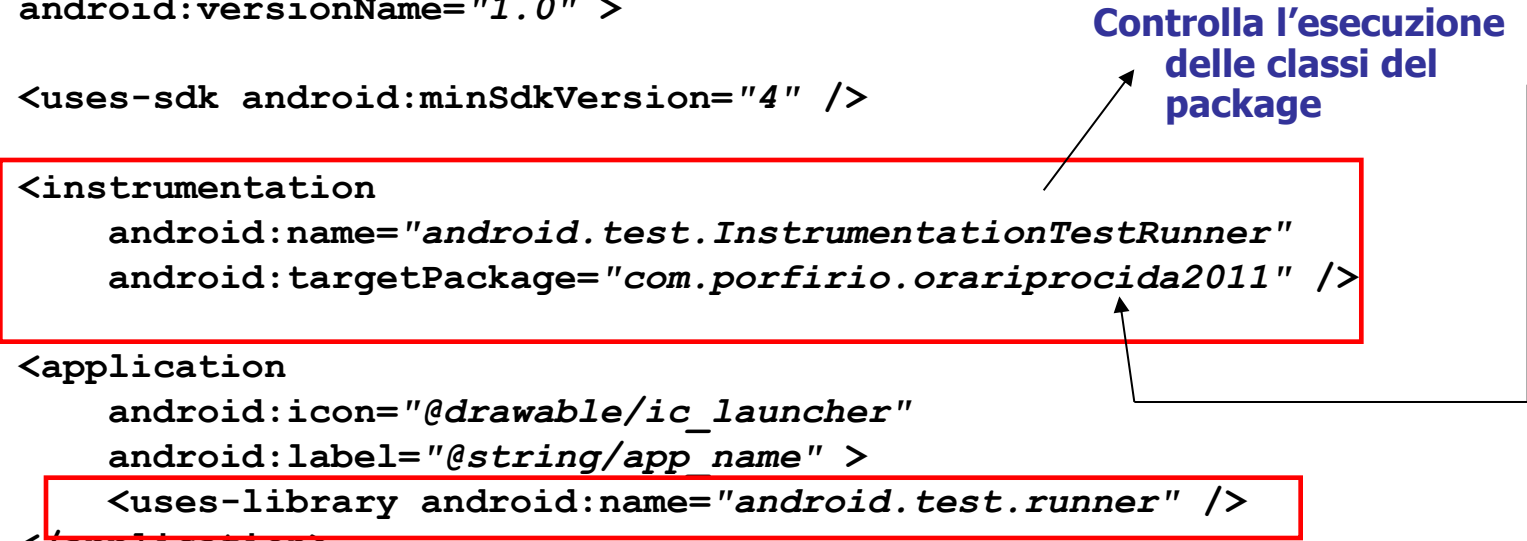
  <uses-sdk android:minSdkVersion="4" />

  <instrumentation
    android:name="android.test.InstrumentationTestRunner"
    android:targetPackage="com.porfirio.orariprocida2011" />

  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name" >
    <uses-library android:name="android.test.runner" />
  </application>

</manifest>
```

Controlla l'esecuzione delle classi del package



Creazione di una classe di test (ADT)

http://developer.android.com/tools/testing/testing_eclipse.html

(in Android Studio non sono ancora presenti appositi wizard)

<http://developer.android.com/training/testing/ui-testing/espresso-testing.html>

The screenshot shows the 'New JUnit Test Case' dialog box in the Eclipse IDE. The title bar reads 'New JUnit Test Case'. Inside, the 'JUnit Test Case' section has a warning icon and the text 'Superclass does not exist.' Below this, there are two radio buttons: 'New JUnit 3 test' (selected) and 'New JUnit 4 test'. The 'Source folder:' field contains 'OrariProcida2011Testing/src' with a 'Browse...' button. The 'Package:' field contains 'com.porfirio.orariprocida2011.test' with a 'Browse...' button. The 'Name:' field contains 'OrariProcida2011ActivityTests'. The 'Superclass:' field contains 'android.test.ActivityInstrumentationTestCase2<OrariProcida2011Activity>' with a 'Browse...' button. Under the heading 'Which method stubs would you like to create?', there are four checkboxes: 'setUpBeforeClass()' (unchecked), 'tearDownAfterClass()' (unchecked), 'setUp()' (checked), and 'tearDown()' (checked). There is also an unchecked checkbox for 'constructor'. Below this, a question asks 'Do you want to add comments? (Configure templates and default value [here](#))' with an unchecked checkbox for 'Generate comments'. At the bottom, the 'Class under test:' field contains 'com.porfirio.orariprocida2011.OrariProcida2011Activity' with a 'Browse...' button. The bottom of the dialog has a help icon, a '< Back' button, a 'Next >' button, a 'Finish' button, and a 'Cancel' button.

Metodi generati

```
package com.porfirio.orariprocida2011.test;

import com.porfirio.orariprocida2011.OrariProcida2011Activity;
import android.test.ActivityInstrumentationTestCase2;

public class OrariProcida2011ActivityTests extends
    ActivityInstrumentationTestCase2<OrariProcida2011Activity> {

    public OrariProcida2011ActivityTests() {
        super("com.porfirio.orariprocida2011", OrariProcida2011Activity.class);
    }

    @Override
    protected void setUp() throws Exception {
        super.setUp();
    }

    protected void tearDown() throws Exception {
        super.tearDown();
    }
}
```

Primi test case

Precondizioni: esistenza degli oggetti utilizzati nel test

```
...  
public class OrariProcida2011ActivityTests  
    extends  
    ActivityInstrumentationTestCase2<OrariProci  
        da2011Activity> {  
  
    private OrariProcida2011Activity mActivity;  
    private TextView mTextView1;  
    private ListView mListView;  
  
    @Override  
    protected void setUp() throws Exception {  
        super.setUp();  
  
        mActivity = this.getActivity();  
        mTextView1=(TextView)  
            mActivity.findViewById(R.id.textView1);  
        mListView =(ListView)  
            mActivity.findViewById(R.id.listMezzi);  
    }  
}
```

Oggetti utilizzati nel test

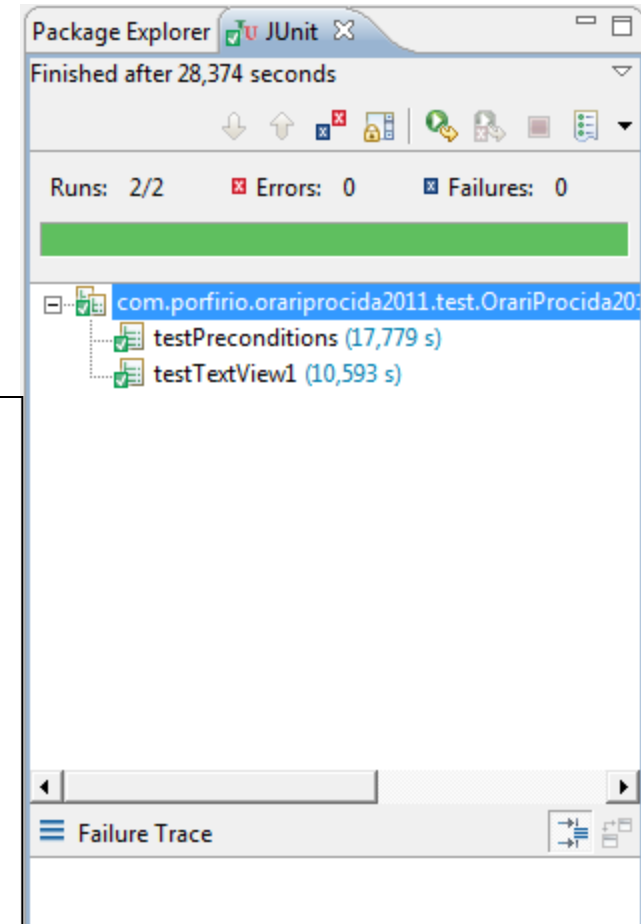
```
public void testPreconditions() {  
    assertNotNull(mTextView1);  
    assertNotNull(mListView);  
}  
  
public void testTextView1() {  
    String resourceString = new  
        String(mActivity.getString(com.porfirio.or  
            ariprocida2011.R.string.mezzo));  
    assertEquals(resourceString, (String)mTextView1.  
        getText());  
}  
  
protected void tearDown() throws Exception  
{  
    super.tearDown();  
}  
}
```

Test sul corretto valore di una casella di testo

Esecuzione dei test

Da Eclipse Run as JUnit Test Sequenza di esecuzione (Console) (idem su Android Studio)

```
[2011-11-16 19:33:36 - OrariProcida2011Testing] -----
[2011-11-16 19:33:36 - OrariProcida2011Testing] Android Launch!
[2011-11-16 19:33:36 - OrariProcida2011Testing] adb is running normally.
[2011-11-16 19:33:36 - OrariProcida2011Testing] Performing
android.test.InstrumentationTestRunner JUnit launch
[2011-11-16 19:33:36 - OrariProcida2011Testing] Automatic Target Mode: using existing emulator
'emulator-5554' running compatible AVD 'AVD_1_6'
[2011-11-16 19:33:36 - OrariProcida2011Testing] Uploading OrariProcida2011Testing.apk onto
device 'emulator-5554'
[2011-11-16 19:33:36 - OrariProcida2011Testing] Installing OrariProcida2011Testing.apk...
[2011-11-16 19:33:40 - OrariProcida2011Testing] Success!
[2011-11-16 19:33:40 - OrariProcida2011Testing] Project dependency found, installing:
OrariProcida2011
[2011-11-16 19:33:44 - OrariProcida2011] Application already deployed. No need to reinstall.
[2011-11-16 19:33:44 - OrariProcida2011Testing] Launching instrumentation
android.test.InstrumentationTestRunner on device emulator-5554
[2011-11-16 19:33:44 - OrariProcida2011Testing] Collecting test information
[2011-11-16 19:33:48 - OrariProcida2011Testing] Sending test information to Eclipse
[2011-11-16 19:33:48 - OrariProcida2011Testing] Running tests...
[2011-11-16 19:34:23 - OrariProcida2011Testing] Test run finished
```



Un test più complesso

Esempio di test che interagisce con l'interfaccia utente, settando un valore di uno Spinner

```
public void testListaVuota() {  
    mActivity.runOnUiThread(  
        new Runnable() {  
            public void run() {  
                mSpnPortoPartenza.setSelection(1);  
                mSpnPortoArrivo.setSelection(1);  
                assertEquals(0, (int)mListView.getCount());  
            }  
        });  
    mInstrumentation.waitForIdleSync();  
}
```

Per settare un campo di una oggetto sulla UI, è necessario, in Android, farlo interagendo nello stesso thread della UI stessa

Per impedire la concorrenza tra eventi dell'utente reale ed eventi simulati da test
`setActivityInitialTouchMode(false);`

Activity Testing

Finora sono stati esemplificati casi di test riguardanti l'interazione con la UI in un'Activity

Altri possibili test da eseguire su di una Activity:

Testing in risposta a eventi riguardanti il ciclo di vita dell'Activity

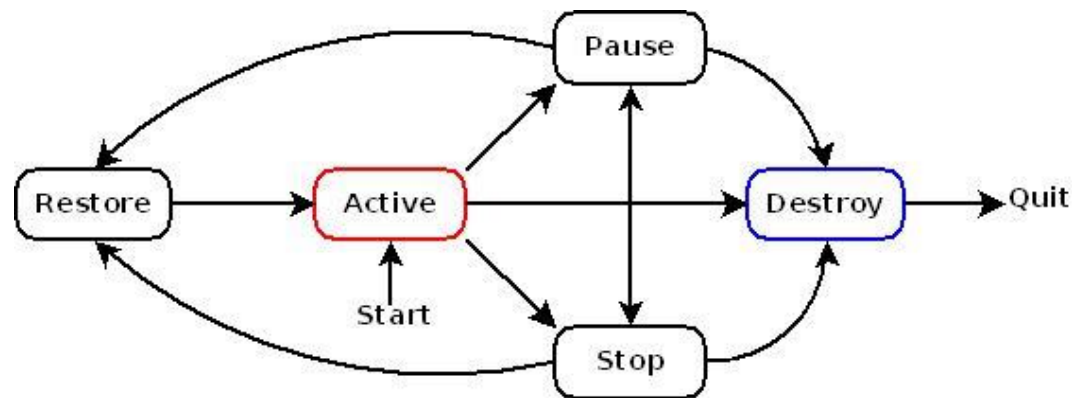
Testing in risposta ad eventi di sistema

Testing in risposta ad eventi provenienti da sensori

Testing del ciclo di vita di un Activity

Per simulare una pausa e resume, ad esempio:

```
Instrumentation mInstr = this.getInstrumentation();  
mInstr.callActivityOnPause(mActivity);  
mInstr.callActivityOnResume(mActivity);
```



Testing in isolamento

Per realizzare Unit Testing è necessario limitare al minimo e controllare le dipendenze dell'unità testata dal resto del software e dell'ambiente di esecuzione

La classe `IsolatedContext` è in grado di riprodurre un contesto di esecuzione fittizio, da utilizzare tutte le volte che sia necessario, senza dipendere dal reale stato del sistema

Per emulare gli eventi di sistema si possono utilizzare le classi del package *android.hardware*

Per emulare I sensori si possono usare le classi *Sensor*, *SensorEvent*, *SensorEventListener* e *SensorManager*, ridefinendole in modo che generino eventi fittizi

Esempio Mock

SMSReceiver è un Broadcast Receiver che ascolta per la ricezione di SMS

SMS è una Activity che viene avviata da SMSReceiver in seguito alla ricezione di un SMS e ne visualizza il testo

MockProvider è una classe che estende Thread e, tramite Intent si dichiara (tramite Intent) in grado di inviare SMS e controllarne il delivery

SMSMock è una classe che estende SMS, imitandolo. In pratica definisce e avvia un MockProvider

SMSTesting è una classe di test che, così come SMSMock, definisce e avvia un MockProvider e gli chiede di inviare un messaggio, come prova

Class Diagram

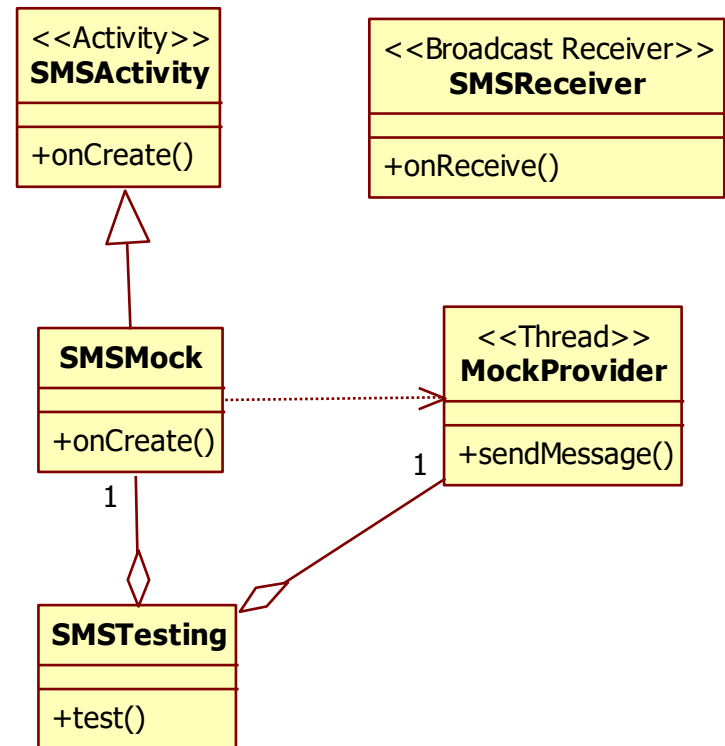
SMSTesting istanzia **SMSMock** e **MockProvider**

SMSMock sostituisce il metodo **onCreate** di **SMSActivity** e, istanziandosi, avvia **MockProvider** e lo dichiara come gestore degli SMS (al posto di un reale fornitore)

MockProvider dichiara due Intent corrispondenti a eventi di Invio e Consegna del messaggio

SMSTesting chiede a **MockProvider** di "inviare" un messaggio

SMSReceiver riceve i messaggi fittiziamente inviati da **MockProvider** e li gestisce come se fossero reali



Codice

```
...

public class SmsTesting extends ActivityInstrumentationTestCase2<SMSMock> {
    private SMSMock myActivity;
    private MockProvider mymockprov;
    ...
    @Override
    protected void setUp() throws Exception{
        super.setUp();
        setActivityInitialTouchMode(false);
        mymockprov = new MockProvider(SmsManager.getDefault(),getInstrumentation().getContext());
    }
    public void testcase1(){mymockprov.invia_messaggio(phoneNumber,messaggio);}

...

public class MockProvider extends Thread{
    private Context ctx; private SmsManager sms;
    private PendingIntent sentPI; private PendingIntent deliveredPI;
    ...
    @Override
    public void run() {
        sentPI = PendingIntent.getBroadcast(ctx, 0, new Intent(SENT), 0);
        deliveredPI = PendingIntent.getBroadcast(ctx, 0, new Intent(DELIVERED), 0);
    }
    public void invia_messaggio(String PHNUM, String MEX){sms.sendTextMessage(PHNUM, null,MEX, sentPI,
        deliveredPI);}
}
```

Unit Testing di altri componenti

Per testare il ciclo di vita di un Service si possono utilizzare i metodi *Context.startService* e *Context.bindService*

Il testing di un servizio è più semplice del testing di una Activity, perchè non dipende da eventi utente e di sistema

Un Broadcast Receiver è molto semplice da testare. Per avviarlo da test si può utilizzare il metodo *Context.sendBroadcast* per simulare l'invio di un Intent

Un ContentProvider fornisce un'astrazione di accesso ai dati. Deve essere testato rispetto all'interfaccia di accesso che fornisce

Alcune apposite classi da cui ereditare

ServiceTestCase, ProviderTestCase2

Livelli di Testing

Dopo lo Unit Testing:

Integration Testing

Le varie unità sono testate in insiemi più grandi

System Testing in tre stage

Simulation stage

Il software viene testato in isolamento in un ambiente completamente simulato

Prototyping stage

L'ambiente reale inizia a rimpiazzare quello simulato, ma il software è eseguito sempre su di un emulatore

Pre-production stage

Il software è eseguito su di un device reale in un ambiente reale

DDMS

Il DDMS (Dalvik Debug Monitor Server) è uno strumento dell'Android SDK particolarmente utile in fase di prototyping

Monitora il comportamento della macchina virtuale

Accesso al file system

Thread in esecuzione

Allocazione della memoria

Log dei messaggi

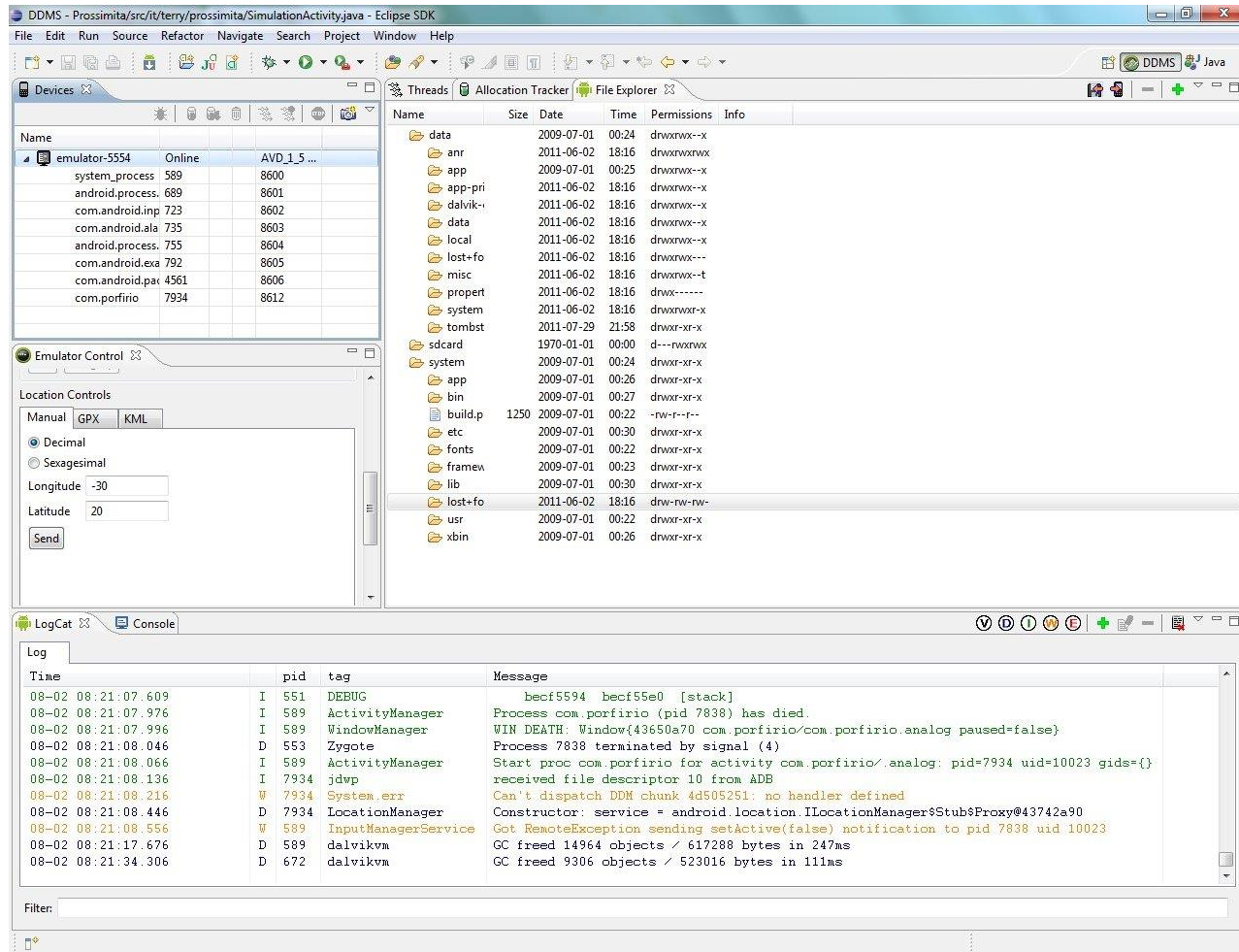
Consente l'emulazione (spoofing) di

Variazioni nelle coordinate GPS

Ricezioni di messaggi SMS

Ricezione di telefonate

DDMS



DDMS è un eseguibile nell'Android SDK, ma anche una perspective in Eclipse ADT.

In Android Studio, è chiamato Android Device Monitor (menu Tools)

Robotium

Robotium è un framework a supporto del testing di unità delle Activity che estende e potenzia Junit. In particolare:

è più semplice scrivere test che riguardano più Activity, Dialog, Toast, Menu e Context Menu.

E' migliorata la leggibilità dei test case

I test case sono meno dipendenti dalla variabilità dei tempi di esecuzione

Robotium è un progetto open source la cui prima versione è stata rilasciata a gennaio 2010

<http://code.google.com/p/robotium/>

Caratteristiche di Robotium

Il funzionamento di Robotium è tutto basato sull'utilizzo di un oggetto denominato SOLO

```
Solo solo = new Solo(getInstrumentation(),getActivity());
```

Tramite l'oggetto solo è possibile interrogare e modificare i widget della UI, eventualmente anche senza conoscerne l'identificativo

Particolarmente utile nei test di accettazione

Ad esempio, è possibile selezionare l'insieme dei widget visibili oppure è possibile selezionare un widget in base al testo che mostra

Esempi

```
public void testTextView(){
    String resourceString = new
String(solo.getString(com.porfirio.orariprocida2011.R.string.mezzo)
);
    TextView mTextView1=solo.getText(1);
    assertEquals(resourceString, (String)mTextView1.getText());
}
```

```
public void testButtonRobotium(){
TextView mTxtOrario=solo.getText(3);
String initial=new String(mTxtOrario.getText().toString());
    solo.clickOnButton("<<");
    solo.clickOnButton(">>");
    assertEquals(mTxtOrario.getText().toString(), initial);
}
```

Android Espresso

<https://code.google.com/p/android-test-kit/wiki/Espresso>

Framework Android per test a partire dalla UI

Alternativo a Robotium

Prodotto da Google

Integrato con ADT e Android Studio

Utilizzabile per tutte le versioni Android a partire dalla 2.3.3 (android 10)

Funziona nell'ambito di Android test unit (come Robotium)

Android Espresso: alcuni dettagli

Il metodo onView consente di ottenere un riferimento ad un oggetto della Gui

I metodi with* consentono di individuare un oggetto della GUI (ad esempio withID)

Il metodo perform consente di eseguire un evento (ad esempio perform(click()))

Il metodo scrollTo risolve il problema di assicurarsi della visibilità di un particolare elemento

Il metodo check() aiuta a scrivere opportune asserzioni

Genera molte utili informazioni sia in caso di fallimento che di test con successo (verso logcat)

Strumenti di Capture & Replay

- **Esistono alcuni strumenti (correntemente in sviluppo) che supportono il Capture & Replay di casi di test per applicazioni Android e la loro traduzione in test Junit**
 - TestDroid (a pagamento, per ADT, genera test che utilizzano Robotium)
<http://testdroid.com/products>
 - Robotium Recorder (gratuito per i primi 5 test generati, sia per ADT che per Android Studio, fa parte del progetto Robotium)
<http://robotium.com/products/robotium-recorder>
 - Espresso Test Recorder (gratuito, integrato con le versioni della distribuzione sperimentale Canary di Android Studio)

Robotium Recorder

Robotium Recorder è disponibile, in versione gratuita, all'indirizzo:

<http://robotium.com/pages/free-trial>

Anche Robotium Recorder è disponibile sotto forma di estensione di Eclipse, scaricabile da:

<http://recorder.robotium.com/updates>

E' possibile, però, catturare (e salvare come test Junit) solo 5 casi di test

Monkey

Monkey è un'utility interna fornita con l'android SDK, che è in grado di generare eventi utente pseudocasuali su una qualsiasi interfaccia, registrando gli eventuali crash

Monkey gira all'interno del dispositivo; per avviarla bisogna passare per adb. Ad esempio, da linea di comando:

```
adb shell monkey -v -p  
com.porfirio.orariprocida2011 30
```

Output di Monkey

```
:Monkey: seed=0 count=30
:AllowPackage: com.porfirio.orariprocida2011
:IncludeCategory: android.intent.category.LAUNCHER
:IncludeCategory: android.intent.category.MONKEY

// Event percentages:
// 0: 15.0%
// 1: 10.0%
// 2: 15.0%
// 3: 25.0%
// 4: 15.0%
// 5: 2.0%
// 6: 2.0%
// 7: 1.0%
// 8: 15.0%

:Switch:
  #Intent;action=android.intent.action.MAIN;category=android.intent.category.LAUNCHER;launchFlags=0x10000000;component=com.porfirio.orariprocida2011/.OrariProcida2011Activity;end

  // Allowing start of Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER]
  cmp=com.porfirio.orariprocida2011/.OrariProcida2011Activity } in package com.porfirio.orariprocida2011

:Sending Pointer ACTION_MOVE x=-4.0 y=2.0
:Sending Pointer ACTION_UP x=0.0 y=0.0
:Sending Pointer ACTION_DOWN x=47.0 y=122.0

Events injected: 30

:Dropped: keys=0 pointers=0 trackballs=0 flips=0

## Network stats: elapsed time=7766ms (7766ms mobile, 0ms wifi, 0ms not connecte

d)

// Monkey finished
```

Monkeyrunner

Monkeyrunner, a differenza di monkey, è un API che consente la scrittura di programmi in grado di controllare un dispositivo Android dall'esterno

Ad esempio è possibile scrivere un programma Python che installa un'applicazione, esegue casi di test, invia eventi, salva screenshot

Esempio di programma:

```
from com.android.monkeyrunner import MonkeyRunner, MonkeyDevice
device = MonkeyRunner.waitForConnection()
device.installPackage('myproject/bin/MyApplication.apk')
package = 'com.example.android.myapplication'
activity = 'com.example.android.myapplication.MainActivity'
runComponent = package + '/' + activity
device.startActivity(component=runComponent)
device.press('KEYCODE_MENU', MonkeyDevice.DOWN_AND_UP)
result = device.takeSnapshot()
result.writeToFile('myproject/shot1.png', 'png')
```

Confronti

- **I test scritti con Robotium, Espresso, ... sono scritti in Java e agiscono a livello di virtual machine**
- **I test eseguiti da Monkey e quelli scritti con MonkeyRunner agiscono a livello di stream degli eventi e sono eseguiti a livello di shell del sistema operativo**
 - In alternativa, è possibile interagire direttamente con lo stream degli eventi utilizzando le primitive *getevent* e *sendevent*

Android Ripper

Una alternativa, con più “intelligenza” di Monkey è l’Android Ripper sviluppato all’Università di Napoli

Navigazione “in ampiezza” o “in profondità”
dell’interfaccia utente di un’applicazione Android

Esecuzione di eventi su tutti i Widget trovati

Generazione di sequenze di esecuzione

Generazione di casi di test JUnit

Rilevazione automatica di crash

Valutazione della copertura ottenuta (con Emma)

Ripping Algorithm

1. Task List Initialization;



4. Execute the Task;

5. Abstract the Current GUI Abstract State;

6. Update the GUI Tree Model;

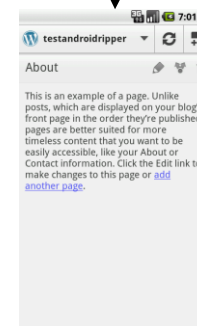
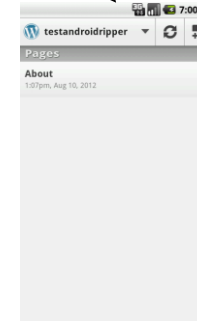
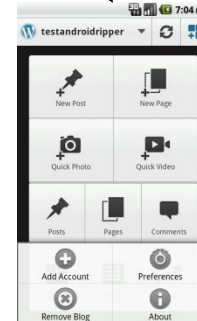
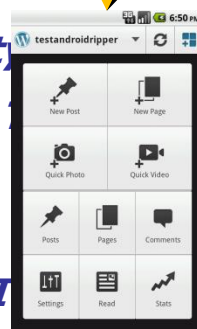
7. if (GUI Exploration Criterion)

8. Define New Tasks ;

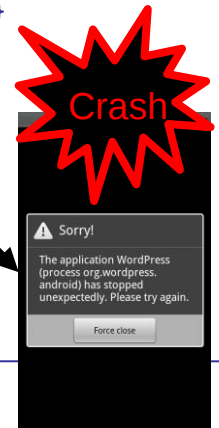
9. Add New Tasks To The Task List;

10.

11. }



37



Parametri

- **L'esempio precedente mostra:**
 - Navigazione in ampiezza dell'albero delle interfacce utente
 - Estrazione di tutti i widget presenti in ogni interfaccia
 - Inserimento di numeri interi casuali nei campi di testo
 - Navigazione di tutte le interfacce che abbiano almeno un widget diverso dalle interfacce già visitate
 - Terminazione della visita al raggiungimento di foglie tutte corrispondenti a stati già visitati

Configurazione

- **Il Ripper può essere configurato in base a:**
 - tipo di esplorazione
 - *Es.: random o sistematica*
 - strategia di navigazione
 - *Es.: casuale, in ampiezza o in profondità*
 - insieme di widget considerati
 - *Es.: tutti i widget per i quali sia stato registrato un listener*
 - Valori da inserire nei campi di testo
 - *Es.: numeri interi casuali o costanti*
 - Criterio di equivalenza tra schermate
 - *Es.: considerare equivalenti (e non esplorare ulteriormente) interfacce che abbiano almeno un widget diverso dalle interfacce già visitate, oppure che abbiano almeno un listener diverso, oppure che differiscano per almeno un valore in un campo di input*
 - Criterio di Terminazione
 - *Ad esempio quando tutte le foglie sono state visitate, oppure al raggiungimento di una profondità massimo, di un numero di stati prefissato oppure di un tempo massimo prefissato*

Osservazioni e Problemi

- La navigazione sistematica può essere meno efficace di quella casuale
- La navigazione casuale è molto meno efficiente di quella sistematica
- Un criterio di equivalenza troppo discriminante (ad esempio che consideri diverse due schermate che differiscono solo per il valore di un'etichetta) può portare a delle navigazioni indefinitamente lunghe (ad esempio se il valore dell'etichetta è l'istante attuale)
- Un criterio di equivalenza poco discriminante può avere scarsa efficacia (scoprire poche interfacce diverse)
- Non tutte le precondizioni possono essere controllate (ad es. il valore di risorse remote, come in una applicazioni di scommesse, oppure l'orario esatto, in una applicazione orologio), quindi alcuni test generati potrebbero non essere ripetibili
- Alcuni widget (ad esempio le liste) possono essere composte di un numero di widget componenti (gli elementi della lista) indefinitamente crescente (in tal caso sarebbe più saggio non considerare tutti questi widget, ma solo un quantitativo limitato di essi, ad esempio i primi tre)

Proposte di progetto

- **Confrontare Android Ripper con altri strumenti analoghi di generazione di casi di test per applicazioni Android**
 - Ad esempio, Robo di Google
- **Modificare Android Ripper proponendo, realizzando e provando diverse tecniche di esplorazione / generazione valori / valutazione equivalenza**

Emma

Emma è uno strumento di strumentazione del codice sorgente che consente di valutare l'effettiva copertura del codice ottenuta a seguito dell'esecuzione di un insieme di casi di test

Può essere eseguito solo da linea di comando, tramite adb. Ad esempio:

```
adb shell am instrument -w -e coverage true [Package di  
test]/android.test.InstrumentationTestRunner
```

Genera report e metriche, anche in formato HTML

Molto utile per il testing White Box

Output di EMMA

EMMA Coverage Report (generated Wed Nov 09 20:01:15 CET 2011)

[all classes]

OVERALL COVERAGE SUMMARY

name	class, %	method, %	block, %	line, %
all classes	52% (14/27)	40% (29/73)	67% (3159/4746)	51% (275,9/545)

OVERALL STATS SUMMARY

total packages: 1
total executable files: 5
total classes: 27
total methods: 73
total executable lines: 545

COVERAGE BREAKDOWN BY PACKAGE

name	class, %	method, %	block, %	line, %
com.porfirio.orariprocida2011	52% (14/27)	40% (29/73)	67% (3159/4746)	51% (275,9/545)

[all classes]

EMMA 0.0.0 (unsupported private build) (C) Vladimir Roubtsov

Copertura del codice con EMMA

```
118     @Override
119     public void onCreate(Bundle savedInstanceState) {
120         super.onCreate(savedInstanceState);
121         Log.d("ACTIVITY", "create");
122         setContentView(R.layout.main);
123
124         myManager = (LocationManager) getSystemService(LOCATION_SERVICE);
125         criteria = new Criteria();
126         criteria.setPowerRequirement(Criteria.POWER_LOW);
127         criteria.setAccuracy(Criteria.ACCURACY_COARSE);
128         BestProvider = myManager.getBestProvider(criteria, true);
129
130         AlertDialog.Builder builder = new AlertDialog.Builder(this);
131         builder.setMessage("Gli orari sono quelli resi noti dalle compagnie di n
132             .setCancelable(false)
133             .setPositiveButton("OK", new DialogInterface.OnClickListener() {
134                 public void onClick(DialogInterface dialog, int id) {
135                     dialog.cancel();
136                 }
137             });
138         aboutDialog = builder.create();
139
```

Google Firebase

Firestore è una soluzione cloud acquisita nel 2014 da Google che raccoglie in sé diversi servizi cloud offerti gratuiti o a pagamento

Firestore può essere utilizzato direttamente dalla sua interfaccia web oppure integrato in Android Studio

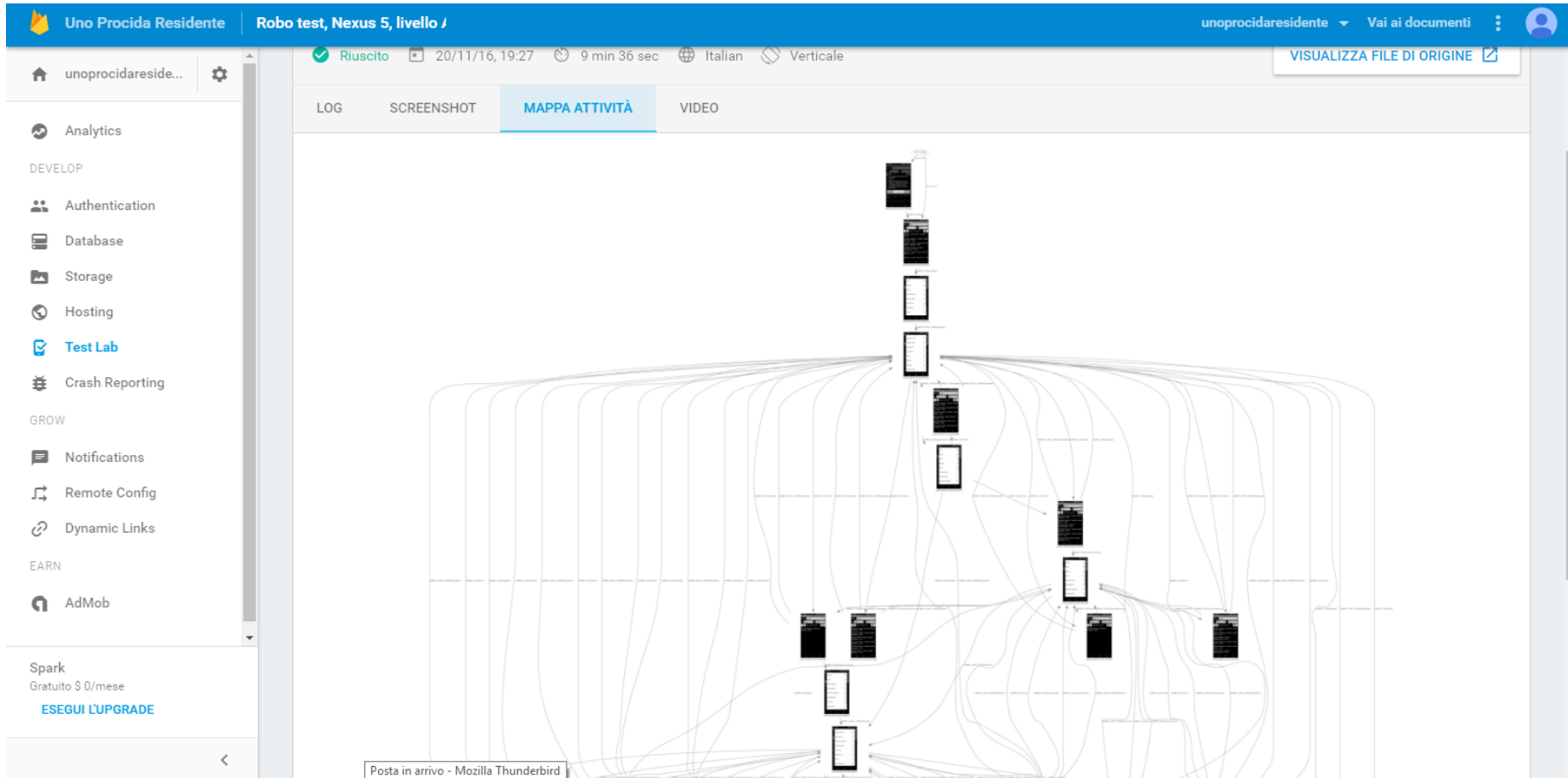


https://firebase.googleblog.com/2016/11/live-from-firebase-dev-summit-in-berlin.html?utm_campaign=Firebase_event_awareness_en_09-19-16_&utm_source=fgc&utm_medium=banner

Firebase Test Lab

- **In particolare Firebase Test Lab consente di eseguire automaticamente, in ambiente cloud:**
 - Test scritti con Android Espresso
 - Test generati automaticamente da uno strumento chiamato Robo
 - *Robo cerca di eseguire sistematicamente azioni avendo osservato i possibili widget sui quali applicarle*
 - Firebase Test Lab genera, screenshot e video delle esecuzioni fatte, le organizza in una mappa delle navigazioni, e riporta il logcat con le eccezioni ricevute

Esempio Firebase



Progetto proposto

- **Provare e confrontare in termini di funzionalità e prezzi le risorse cloud a supporto di attività di testing automation**
 - a partire da Firebase, TestDroid, ...

Confronto tra strumenti

- **Recentemente, è stato realizzato uno studio sperimentale di confronto tra diversi strumenti per la generazione ed esecuzione automatica di test per applicazioni Android**
- **I vari strumenti (non tutti quelli esistenti) sono stati installati nell'ambito di una stessa macchina Linux e provati su di uno stesso insieme di applicazioni**

- **Shauvik Roy Choudhary, Alessandra Gorla, Alessandro Orso, Automated Test Input Generation for Android: Are We There Yet?**
 - <https://arxiv.org/abs/1503.07217>

Progetto proposto

- **Installare, utilizzare ed estendere lo strumento proposto, aggiungendo prove effettuate con altri strumenti (ad es. Android Ripper) o su altre applicazioni**

Appendice

Esecuzione di Android Ripper

Prerequisites

Android Ripper has been tested on Windows 10, Linux Ubuntu and Linux Mint.

- Android SDK
- Oracle Java 8
- Use Android SDK Manager to download and install the latest:
 - Android SDK Tools
 - Android SDK Platform Tools
 - Android SDK Build Tools
- Use Android SDK Manager to download and install for each required Android API:
 - the SDK Platform
 - the Intel and ARM System Images

-Add to the PATH environment variable the following directories:

PathToAndroidSdk/tools/

PathToAndroidSdk/platform-tools/

PathToAndroidSdk/build-tools/BUILDNUMBER/

(it is variable between different versions of Android SDK)

-Set the following Environment Variables:

JAVA_HOME=PathToJre or PathToJdk

ANDROID_HOME=PathToAndroidSdk

Configurazione di Android Ripper

Le impostazioni sono tutte contenute in un file di configurazione (ad esempio default. properties).

Ad esempio:

```
#Name of the avd equipped with an x86 architecture
avd_name_x86 = testing

#Name of the avd equipped with an ARM architecture
avd_name_arm = testingARM

#Port number used to communicate with the avd
avd_port = 5554

#Exploration strategy. Valid strategies: random, systematic
driver=random

#Number of events to be fired (used only for the random strategy)
events=20

#Time between events (milliseconds)
sleep_after_event=1000

#Strategy implemented by the component that schedule the next event to be executed
#For a Random Exploration Strategy the only possible value is "random" (default)
#For a Systematic Exploration Strategy: "breadth", "depth"
scheduler=random

#Random Seed (used only for the random strategy)
#Default = System.currentTimeMillis()
seed=0
```

Esecuzione di Android Ripper

From shell command line:

```
java -jar AndroidRipper.jar PathToApk  
[PathToConfigurationFile]
```

The default.properties configuration file is used if the last argument is blank. This file is located in the AndroidRipper folder.

A shortcut is given by the Android RipperGUI.jar interactive application

Appendice

Istruzioni per l'esercizio di testing con Robotium Recorder 1/2

La prima parte dell'esercizio si svolge sulla macchina di test approntata presso il Laboratorio di Ingegneria del Software

1. Accedere alla macchina (fisicamente o virtualmente)
2. Avviare la macchina virtuale Robotium Recorder (se non è già avviata)
3. Avviare Eclipse (se non è già avviato)
4. Avviare un emulatore chiamato test (Api Level 15) (se non è già avviato)

Istruzioni per l'esercizio di testing con Robotium Recorder 2/2

5. Per ogni applicazione da testare
 1. *Importare il progetto con l'applicazione (se non già presente)*
 2. *Dal menu contestuale scegliere Robotium Recorder/New Robotium Test*
 3. *Scegliere l'applicazione e dare un nome al test*
 4. *Avviare New Robotium Test*
 5. *Operare sull'emulatore interagendo con l'applicazione sotto test*
 6. *Al termine delle operazioni, premere Stop Robotium Test*
 7. *Premere Save*

Progetto di test e riesecuzione

In questo modo verrà così creato un progetto di test che potrà essere subito rieseguito e, successivamente, modificato

Per rieseguire il test è necessario azzerare i dati dell'applicazione con Clear Data dall'applicazione Settings/Apps sull'emulatore

Duplicare il progetto di test prima di modificarlo per poter tenere nota di tutta la sua evoluzione

E' possibile calcolare la copertura in maniera completamente analoga al caso precedente

*L'unica differenza consiste nello specificare
`android-15` anziché `android-10` come target*

Per poter continuare il proprio lavoro a casa, è opportuno portare una memoria usb con almeno 5 GB liberi che conterrà la macchina virtuale (in formato Oracle VM Virtual Box) sulla quale si è condotta la prima parte dell'esperimento

Robotium Recorder – un esempio completo

Consideriamo un'applicazione molto semplice come esempio

Simply Do è una semplice applicazione Android che consente di gestire azioni da svolgere, organizzate in liste.

Dall'interfaccia principale è possibile:

- vedere l'elenco di tutte le liste;

- aggiungere una nuova lista (semplicemente digitandone il nome e chiedendo di aggiungerla).

Selezionando una lista, è possibile:

- visualizzare tutte le azioni all'interno della lista

- aggiungere un'azione all'interno della lista (scrivendone il nome)

- segnare un'azione come già svolta (selezionandola): in questo caso l'azione sarà visualizzata in grigio e barrata. E' possibile segnare nuovamente un'azione come non ancora svolta selezionandola ulteriormente

Dal menu è inoltre possibile eliminare tutte le azioni già svolte e ordinare le azioni secondo l'ordine prestabilito.

E' possibile personalizzare l'applicazione tramite alcune voci disponibili dal menu (Settings). In particolare è possibile:

- Scegliere l'ordine di visualizzazione delle azioni (prima quelle non ancora effettuate – Active, oppure prima quelle evidenziate – Starred)

- Scegliere l'ordine di visualizzazione delle liste (alfabetico oppure lasciarle in ordine di immissione)

- Decidere se chiedere conferma ogni volta che si tenta di cancellare le azioni già effettuate

- Effettuare il backup sulla memoria locale

- Ripristinare le note salvate in uno dei backup precedenti.

Robotium Recorder – un esercizio

Testare l'applicazione Simply Do eseguendo un insieme di esecuzioni

Nell'ambito di Robotium Recorder

Generando automaticamente al termine i corrispondenti test JUnit
rieseguibili

Cercando di provare l'applicazione in tutti i modi possibili, sulla base dei requisiti e delle strategie di progettazione dei casi di test note

Al termine verranno misurate (con Emma) e valutate le coperture raggiunte, per avere un'idea dell'efficacia dei test generati

Si cerchi di tenere conto del tempo impiegato per eseguire i vari passi del processo

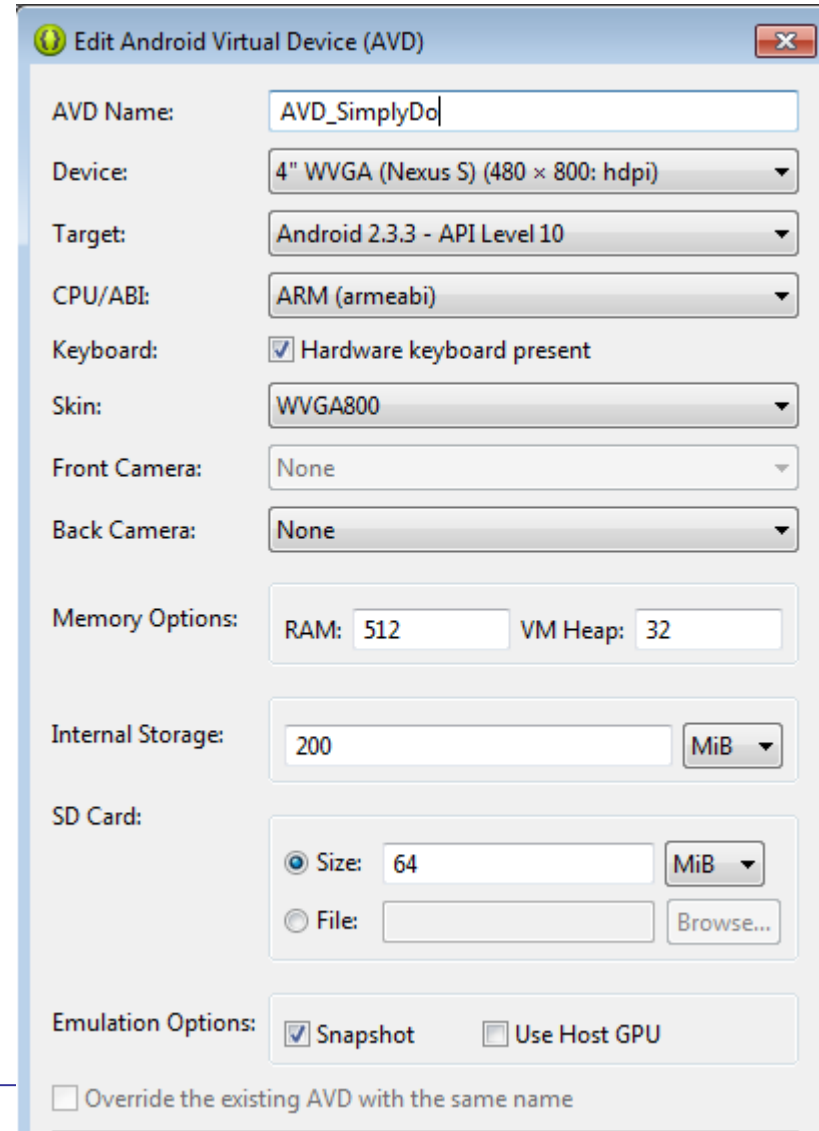
Valutazione dell'efficacia del test

**Come confronto rispetto a Simply Do,
consideriamo la copertura che ho ottenuto
io con una sessione di test durata 8 minuti**

EMMA Coverage Report (generated Mon Nov 24 20:54:43 CET 2014)				
[all classes]				
OVERALL COVERAGE SUMMARY				
name	class, %	method, %	block, %	line, %
all classes	96% (44/46)	65% (161/246)	57% (3153/5523)	55% (708,8/1281)
OVERALL STATS SUMMARY				
total packages:	1			
total executable files:	15			
total classes:	46			
total methods:	246			
total executable lines:	1281			

Istruzioni per l'esercizio di testing con Robotium Recorder e ADT 1/2

- 1. Installare Simply Do scompattando il .zip ed importandolo come progetto Eclipse**
- 2. Creare una macchina virtuale con le caratteristiche in figura e avviarla**
- 3. Registrare una serie di esecuzioni con Robotium Recorder sull'emulatore**



Istruzioni per l'esercizio di testing con Robotium Recorder 2/2

Far generare automaticamente i test case a Robotium Recorder

Rieseguire i casi di test per controllare se la loro esecuzione va a buon fine

Problemi nella riesecuzione potrebbero essere dovuti a click troppo veloci (Robotium Recorder potrebbe avere reazioni lente, talvolta: meglio testare con calma)

Eventualmente, è possibile valutare piccoli difetti nel test generato e ripararli eseguendo il debug del caso di test

Misurare la copertura (vedi istruzioni)

Misurare la copertura (dell'esercizio con Robotium)

1) Avviare l'emulatore

2) Disinstallare SimplyDo

3) Spostarsi nella cartella del sorgente della applicazione da testare (\SimplyDo)

Sotto windows, è preferibile che le cartelle non abbiano spazi

4) [percorso di ant\bin]\ant clean

ant può essere scaricato da <http://archive.apache.org/dist/ant/binaries/apache-ant-1.9.4-bin.zip>

5) [*Percorso di android\tools*]\android update project --path . --target android-10

[*Percorso di android\tools*] potrebbe essere D:\adt-bundle-windows-x86-20140702\sdk\tools

Per generare il file build.xml necessario ad ant

6) Spostarsi nella cartella del caso di test (\SimplyDoTest)

7) [percorso di ant\bin]\ant clean

Misurare la copertura

8) [*Percorso di android|tools*]|android update test-project -p .\ --main ..\SimplyDo

Se SimplyDo si trova in una posizione diversa sul disco al posto di ..\SimplyDo mettere il percorso esatto di SimplyDo (utilizzato al punto 3)

Per creare un file build.xml per il progetto di test collegandolo a quello dell'applicazione

9) Nella cartella del caso di test eseguire :

[percorso di ant\bin]\ant emma debug install test

10) Al termine del caso di test, nella cartella \SimplyDo\bin saranno disponibili i risultati di copertura in vari formati (html, xml, txt)

11) Inviarmi il progetto di test con tutti i file di copertura (è sufficiente zippare le due cartelle SimplyDo e SimplyDoTest)

TestDroid

TestDroid comprende un insieme di strumenti che consentono l'automazione di test di applicazioni Android in ambiente reale e simulato

TestDroid Enterprise

TestDroid Cloud

TestDroid Privateloud

TestDroid Recorder

<http://testdroid.com/>

TestDroid Recorder

TestDroid Recorder è uno strumento, parzialmente di libero utilizzo, che fornisce funzionalità di capture & replay su Android

Capture: è in grado di osservare esecuzione utente su dispositivo reale o anche su emulatore

Replay: è in grado di rieseguire le interazioni catturate e anche di generare codice Junit+Robotium rieseguibile

<http://testdroid.com/product/testdroid-recorder#0>

TestDroid Recorder è disponibile sotto forma di estensione di Eclipse/ADT

<http://www.testdroid.com/updates/>

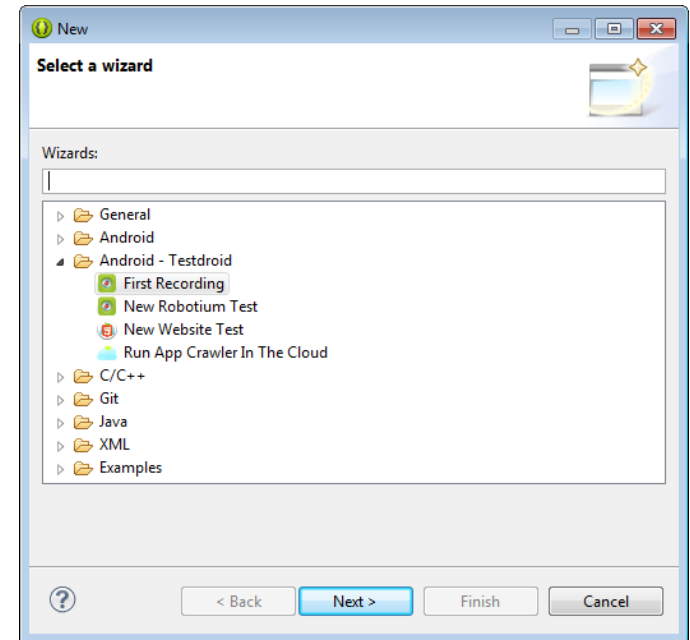
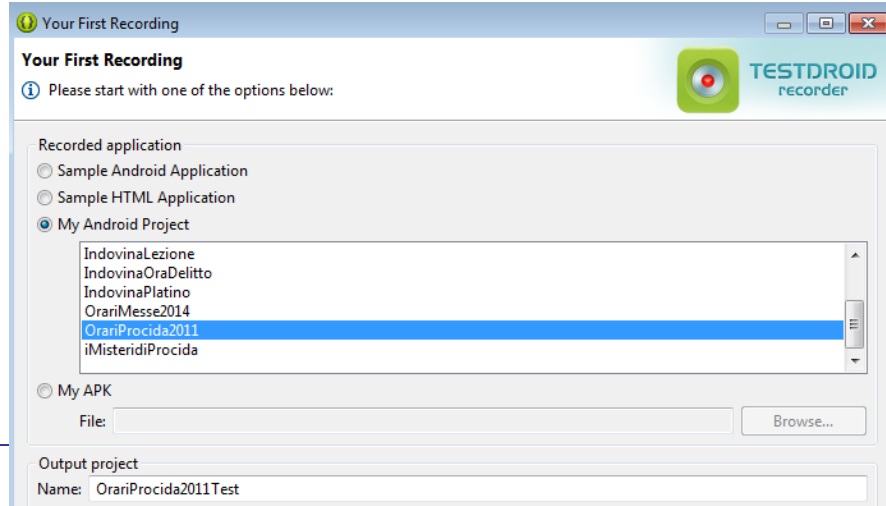
E' necessario registrarsi e scrivere e-mail e password all'atto dell'utilizzo di TestDroid Recorder in Eclipse

TestDroid Recorder

Tutorial per l'utilizzo di TestDroid Recorder sono disponibili a:

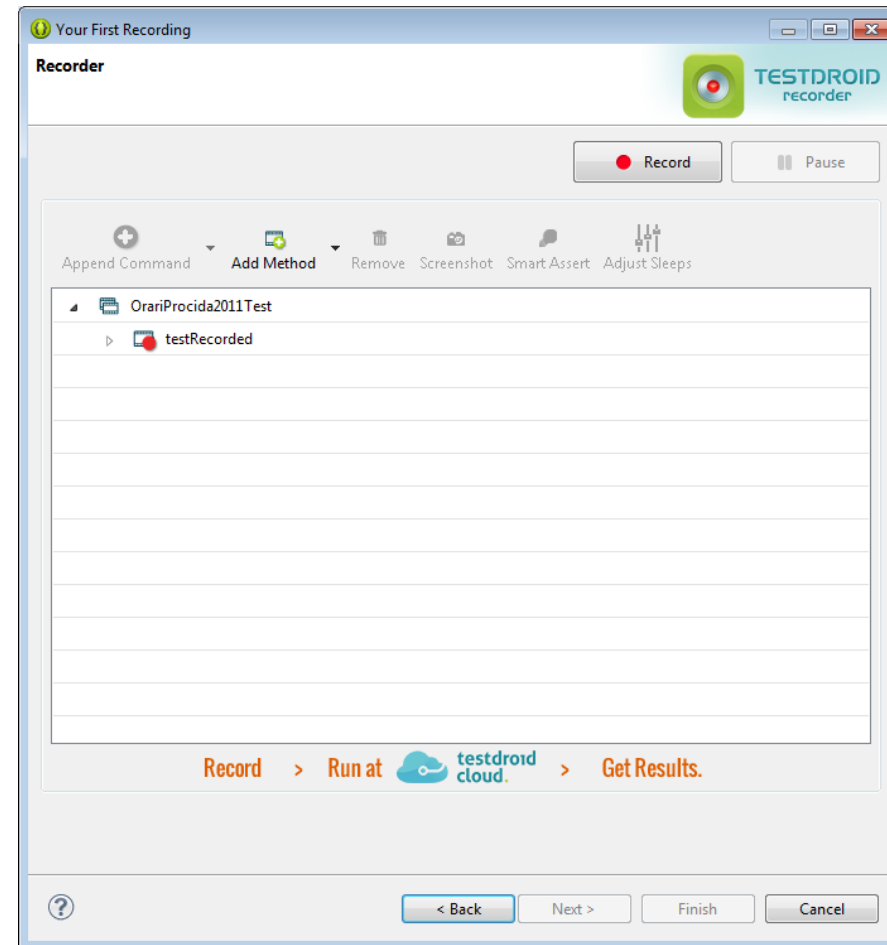
<http://help.testdroid.com/customer/portal/topics/315855-testdroid-recorder-tutorials/articles>

- 1. Avviare New dal menu file o dal menu contestuale dell'app da testare
Scegliere First Recording**
- 2. Se necessario, inserire login e password di registrazione a TestDroid
Selezionare l'app da testare**

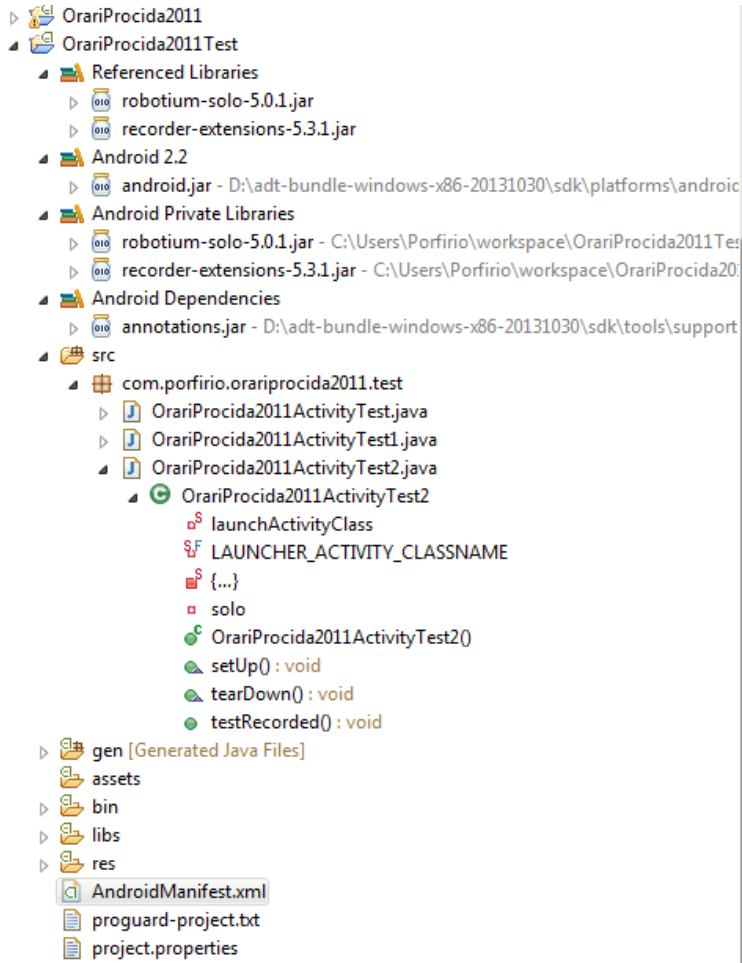


TestDroid Recorder - Tutorial

- 1. Dal pannello è possibile avviare la registrazione premendo Record
L'applicazione verrà installata e avviata sul dispositivo o sull'emulatore scelto**
- 2. Utilizzare l'app eseguendo gli scenari da registrare**
- 3. Premere Stop
I test verranno generati all'interno di un apposito progetto di Test**



TestDroid Recorder - Tutorial



Progetto di test generato

```
...
public void testRecorded() throws Exception {
    try {
        solo.waitForActivity("OrariProcida2011Activity");
        solo.sleep(21500);
        assertTrue("Wait for button (text: OK) failed.",
            solo.waitForButton("OK", 20000));
        solo.clickOnButton("OK");
        solo.sleep(9000);
        assertTrue("Wait for spinner (index: 0) failed.",
            solo.waitForSpinner(0, 20000));
        solo.pressSpinnerItem(0, 1);
        solo.sleep(6700);
        assertTrue(
            "Wait for button (id: com.porfirio.orariprocida2011.R.id.button4)
            failed.",
            solo.waitForButtonById(
                "com.porfirio.orariprocida2011.R.id.button4", 20000));
        solo.clickOnButton((Button) solo
            .findViewById("com.porfirio.orariprocida2011.R.id.button4"));
    } catch (AssertionFailedError e) {
        solo.fail(
            "com.porfirio.orariprocida2011.test.OrariProcida2011ActivityTest2.testRe
            corded_scr_fail",
            e);
        throw e;
    } catch (Exception e) {
        solo.fail(
            "com.porfirio.orariprocida2011.test.OrariProcida2011ActivityTest2.testRe
            corded_scr_fail",
            e);
        throw e;
    }
}
```

TestDroid Recorder - Approfondimenti

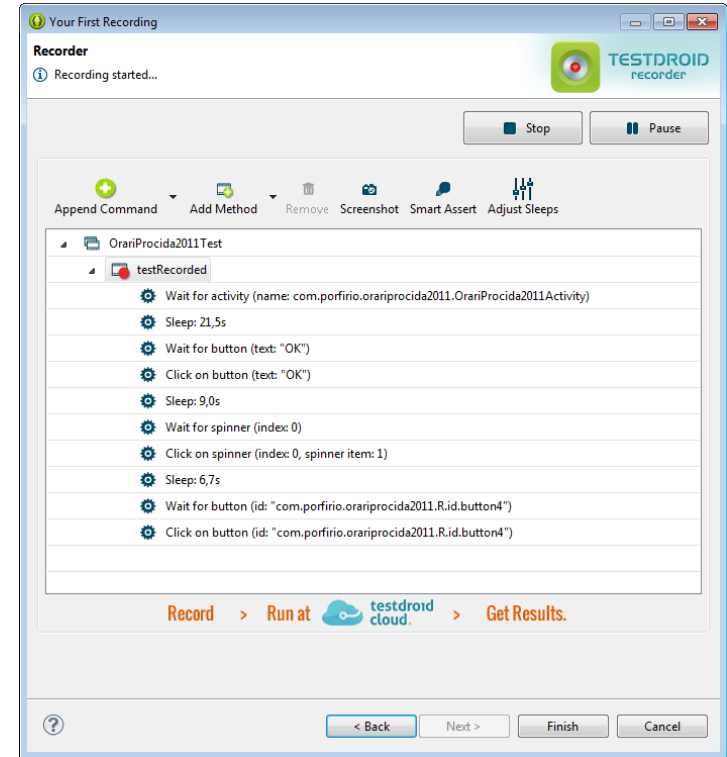
I test generati da TestDroid Recorder possono essere eseguiti automaticamente come qualsiasi altro test, senza ulteriore supporto di TestDroid

Sessioni di Record/Stop ripetute consentono di generare più casi di test

Ad ogni nuovo test l'applicazione viene reinstallata

Le istruzioni Sleep nel caso di test rispecchiano i reali intervalli di tempo tra gli eventi. Possono essere editati sia qui che direttamente nel caso di test

Tempi troppo brevi potrebbero non consentire al sistema di evolvere in tempo prima dell'evento successivo



TestDroid Recorder - Approfondimenti



Tramite Append Command è possibile (dopo il recording e prima della generazione dei casi di test) editare i test aggiungendo comandi specifici oppure asserzioni sui valori dei campi visualizzati

Tramite Add Method è possibile dare un nome ai test prima di registrarli

Tramite Take Screenshot è possibile aggiungere un comando per salvare una schermata grafica all'interno del test stesso

Smart Assert crea automaticamente asserzioni riguardanti la maggior parte delle etichette che riesce a trovare nella schermata a video

Con Adjust Sleeps è possibile variare il tempo di sleep nei test

TestRecorder – un esempio completo

Consideriamo un'applicazione molto semplice come esempio

Simply Do è una semplice applicazione Android che consente di gestire azioni da svolgere, organizzate in liste.

Dall'interfaccia principale è possibile:

vedere l'elenco di tutte le liste;

aggiungere una nuova lista (semplicemente digitandone il nome e chiedendo di aggiungerla).

Selezionando una lista, è possibile:

visualizzare tutte le azioni all'interno della lista

aggiungere un'azione all'interno della lista (scrivendone il nome)

segnare un'azione come già svolta (selezionandola): in questo caso l'azione sarà visualizzata in grigio e barrata. E' possibile segnare nuovamente un'azione come non ancora svolta selezionandola ulteriormente

Dal menu è inoltre possibile eliminare tutte le azioni già svolte e ordinare le azioni secondo l'ordine prestabilito.

E' possibile personalizzare l'applicazione tramite alcune voci disponibili dal menu (Settings). In particolare è possibile:

Scegliere l'ordine di visualizzazione delle azioni (prima quelle non ancora effettuate – Active, oppure prima quelle evidenziate – Starred)

Scegliere l'ordine di visualizzazione delle liste (alfabetico oppure lasciarle in ordine di immissione)

Decidere se chiedere conferma ogni volta che si tenta di cancellare le azioni già effettuate

Effettuare il backup sulla memoria locale

Ripristinare le note salvate in uno dei backup precedenti.

TestRecorder – un esercizio

Testare l'applicazione Simply Do eseguendo un insieme di esecuzioni

Nell'ambito di TestDroid Recorder

Generando automaticamente al termine i corrispondenti test JUnit
rieseguibili

Cercando di provare l'applicazione in tutti i modi possibili, sulla base dei
requisiti e delle strategie di progettazione dei casi di test note

Al termine verranno misurate (con Emma) e valutate le coperture
raggiunte, per avere un'idea dell'efficacia dei test generati

Si cerchi di tenere conto del tempo impiegato per eseguire i vari passi
del processo

Valutazione dell'efficacia del test

**Come confronto rispetto a Simply Do,
consideriamo la copertura che ho ottenuto
io con una sessione di test durata 8 minuti**

EMMA Coverage Report (generated Mon Nov 24 20:54:43 CET 2014)				
[all classes]				
OVERALL COVERAGE SUMMARY				
name	class, %	method, %	block, %	line, %
all classes	96% (44/46)	65% (161/246)	57% (3153/5523)	55% (708,8/1281)
OVERALL STATS SUMMARY				
total packages:	1			
total executable files:	15			
total classes:	46			
total methods:	246			
total executable lines:	1281			

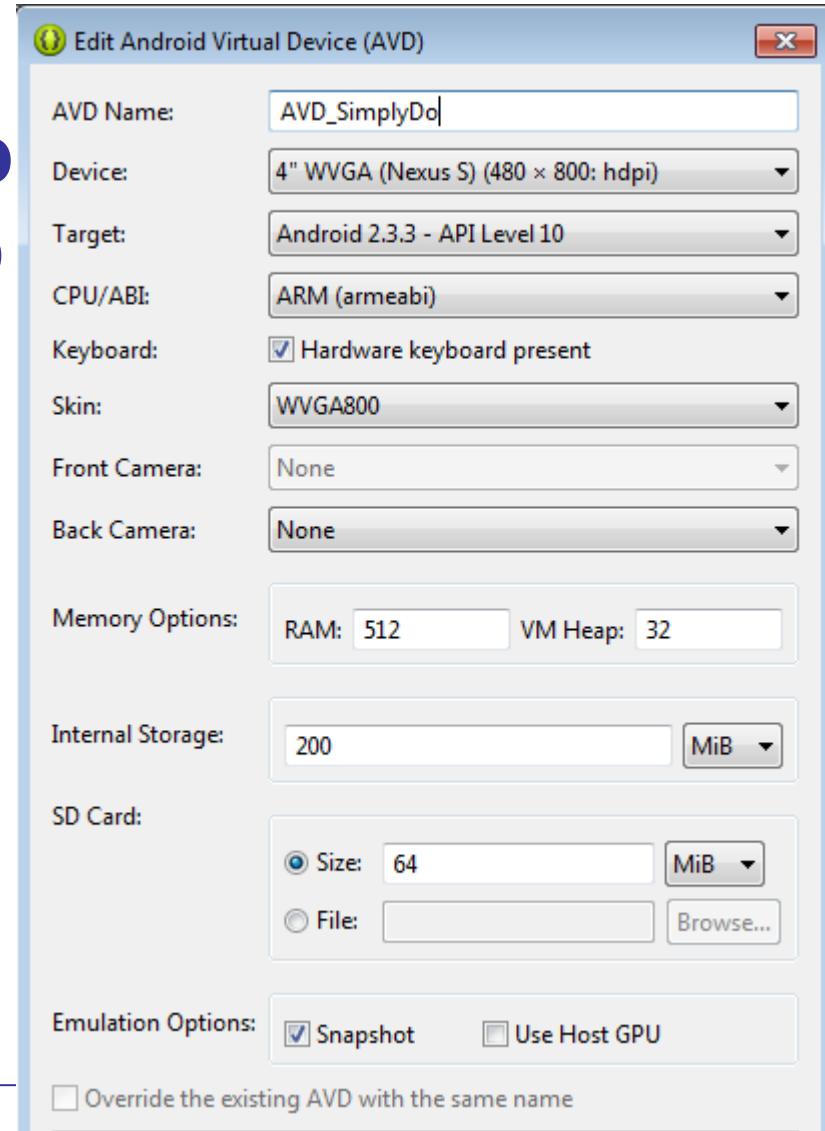
Istruzioni per l'esercizio di testing con TestDroid 1/2

1. Installare Simply Do scompattando il .zip ed importandolo come progetto Eclipse

2. Creare una macchina virtuale con le caratteristiche in figura e avviarla

3. Registrare una serie

Porfirio Tramontana - Android



Istruzioni per l'esercizio di testing con TestDroid 2/2

**Far generare automaticamente i test case a
TestDroid Recorder**

**Rieseguire i casi di test per controllare se la
loro esecuzione va a buon fine**

Problemi nella riesecuzione potrebbero essere dovuti a click troppo veloci (TestDroid Recorder potrebbe avere reazioni lente, talvolta: meglio testare con calma)

Eventualmente, è possibile valutare piccoli difetti nel test generato e ripararli eseguendo il debug del caso di test

Misurare la copertura (vedi istruzioni)

Misurare la copertura (dell'esercizio con TestDroid)

**0) Se non già settato settare JAVA_HOME dal pannello di controllo oppure (ad esempio) con
Set JAVA_HOME=c:\Program Files\Java\jdk1.8.0_25**

1) Avviare l'emulatore

2) Disinstallare SimplyDo

3) Spostarsi nella cartella del sorgente della applicazione da testare (\SimplyDo)

Sotto windows, è preferibile che le cartelle non abbiano spazi

4) [percorso di ant\bin]\ant clean

ant può essere scaricato da <http://it.apache.contactlab.it//ant/binaries/apache-ant-1.9.4-bin.zip>

5) [*Percorso di android\tools*]\android update project --path . --target android-10

[*Percorso di android\tools*] potrebbe essere D:\adt-bundle-windows-x86-20140702\sdk\tools

Per generare il file build.xml necessario ad ant

6) Spostarsi nella cartella del caso di test (\SimplyDoTest)

7) [percorso di ant\bin]\ant clean

Misurare la copertura

8) [*Percorso di android|tools*]\android update test-project -p .\ --main ..\SimplyDo

Se SimplyDo si trova in una posizione diversa sul disco al posto di ..\SimplyDo mettere il percorso esatto di SimplyDo (utilizzato al punto 3)

Per creare un file build.xml per il progetto di test collegandolo a quello dell'applicazione

9) Nella cartella del caso di test eseguire :

[percorso di ant\bin]\ant emma debug install test

10) Al termine del caso di test, nella cartella \SimplyDo\bin saranno disponibili i risultati di copertura in vari formati (html, xml, txt)

11) Inviarmi il progetto di test con tutti i file di copertura (è sufficiente zippare le due cartelle SimplyDo e SimplyDoTest)